*User's Guide*

# *MSPM0 微控制器上的 Zephyr 入门*


**摘要**

本指南为希望通过 MSPM0 微控制器 (MCU) 使用开源 Zephyr 实时操作系统 (RTOS) 的用户提供了一个高级别切入点。其中重点介绍了 Texas Instruments MSPM0 可用的软件资源，说明了 MSPM0 器件的特性，并概述了以下工具：OpenOCD、VSCode 调试及 GNU 调试。

本指南涵盖：

- 什么是 Zephyr
- 为何将 Zephyr 与德州仪器 (TI) MSPM0 微控制器结合使用
- 如何设置 Zephyr 环境
- Zephyr 上游存储库和德州仪器 (TI) 维护的下游存储库之间的差异
- 使用命令行及 Visual Studio Code 调试 Zephyr 工程的概述。
- 如何在 MSPM0 LaunchPad™ 上运行示例


## 内容



## 商标

# 1 什么是 Zephyr？

## 1.1 实时操作系统 (RTOS)

实时操作系统是一种非常适合嵌入式应用程序的计算机操作系统。RTOS 采用小巧设计，具有确定性，开销小，以便在执行命令时对时间敏感。

RTOS 的体量比 Linux 等操作系统轻得多，但仍具有多任务处理、调度和内存处理的诸多优势。凭借这些特性，RTOS 非常适合复杂应用，并且仍然对内存、功耗和计算敏感。

有许多有用资源可用于了解多任务处理、调度和内存架构的复杂性，此处将不对此进行介绍。然而，为了能够启动和运行 Zephyr，重要的是要对以下 RTOS 概念有基本的了解：

- 多任务处理/线程：操作系统采用内核，这是允许多个“用户”或程序访问处理器计算和内存资源的核心进程。这种同时操作是通过线程来完成的，其中每个运行的程序都分配了一个线程（或 Zephyr 中的一个任务）。与调度结合使用时，这些任务可以“同时”运行。
- 调度：内核的核心元件是调度，它允许同时执行的设计。调度器是操作系统内的基本代码块，能够在任务执行期间多次暂停、恢复和切换任务。这意味着，当一个任务有停机时间或没有主动执行代码时，另一个任务可以控制计算和内存，直到完成，或者优先级较高的任务开始。
- 实时操作：在许多应用中，需要对外部激励进行实时响应。这就是 Zephyr 等 RTOS 和 Linux 等典型非实时操作系统之间的区别。创建任务时，会为它们分配优先级，类似于中断。需要时间敏感响应的任务会被分配比非时间敏感任务更高的优先级，这样，它们就可以在创建计算后立即使用计算。

## 1.2 Zephyr 作为开源 RTOS 选项

Zephyr 是一款开源 RTOS，过去十年中在嵌入式开发领域越来越受欢迎。Zephyr 是一个社区维护的 RTOS，无需为许可证支付版税。

# 2 Zephyr 在 MSPM0 上的优势

## 2.1 对比裸机的优势

通常，在单个应用上运行多个进程或任务可能非常具有挑战性，需要软件工程师进行复杂的内存管理。但是，RTOS 通过内核简化了此过程。尽管开销比典型的裸机代码更高，但 Zephyr 等 RTOS 内核由于开销相对较低和编译时内存优化，仍然非常适合内存敏感型应用。Zephyr 的另一个重要考虑因素是代码验证及安全性的易用性。Zephyr 中的所有应用程序代码都在硬件抽象层之上运行，因此验证变得更加简单。软件开发人员知道代码的基础是健全的，只需要用些考虑应用程序代码，而由于 Zephyr 内核提供的额外调试和内核功能，应用程序代码可能更容易调试。

## 2.2 MSPM0 注意事项

MSPM0 系列 Arm Cortex-M0+ 微控制器在功耗、尺寸和成本方面表现出色。因此，MSPM0 适用于大多数每种应用，能够将模拟、通信和辅助控制功能集成到一个封装中。除了 Zephyr 的轻量级和时间敏感特性外，MSPM0 还成为各种嵌入式项目的理想操作系统。

Zephyr 包含以下属性，这些属性有利于功耗、内存并实现 MSPM0 的卓越性能。

- 电源：
  - 空闲操作无需静音，在低功耗模式下可节省能源
  - 在没有任务正在运行时，MCU 可以进入深度睡眠状态
- 内存：
  - Zephyr 的内核可安装在几 KB 的 ROM 和 RAM 中，能够很好地处理 MSPM0 较小的内存资源
- 性能：
  - 抢先式调度器提供了可预测的任务延迟
  - 由于具有严格的实时性能，因此非常适合传感器融合或电机控制等应用
  - 线程/任务处理允许充分利用 CPU 的功能，因为任务仅在必要时运行

总体而言，Zephyr 通过实现基于标准的可扩展 RTOS 环境，增加多任务处理，同时保持器件的效率和确定性行为，补充了 MSPM0 系列器件的低功耗和具有成本效益的优势。

## 2.3 常见应用

通常，工程师会在医疗、工业、汽车和可穿戴应用等时间敏感型应用中使用 RTOS。回顾 Zephyr 的优点可清楚地说明它在这些情况下如何发挥作用。在医疗、工业或汽车应用中，时间敏感属性对于用户安全和应用一致性至关重要，在此类应用中，即时响应至关重要。

Zephyr 栈集成了大量 TI 和第三方传感器，使设计人员无需手动对其进行编码即可快速启动完整系统或原型设计。此外，Zephyr 是开源的，因此会频繁添加更新的传感器。电池充电器、连接解决方案、环境传感器等都可立即接合。

## 2.4 安全概述

在所有这些应用中，无论应用如何，安全性都已成为一个重要的考虑因素。虽然操作系统通过增加的抽象层提供了一个自然应用程序安全层，但 Zephyr 会采取额外措施来提高整个系统的安全性。嵌入式应用中的一个关键安全组件是内存保护，而在裸机应用中，这主要由软件开发人员负责。另一方面，Zephyr 提供了隔离能力，这意味着较小的编码错误不太可能在系统范围内造成后果。然而，这并不意味着 Zephyr 可以提供全面的攻击防护，并且在开发过程中遵循安全编码准则以最大限度减少 Zephyr 内的攻击向量非常重要。虽然这是 Zephyr 安全功能的简要概述，但可以在其文档中找到 Zephyr 安全功能的完整概述。

# 3 如何设置 Zephyr 开发环境

## 3.1 常规设置

Zephyr 的安装方式因用户的操作系统而有所不同。对于本文档，安装将显示在 Ubuntu 22.04 LTS 中。对于 Windows 和 MacOS，请参阅指南。但是，大部分安装使用 Python 和 west 执行的，这意味着操作系统之间安装过程的差异要少得多。

### 3.1.1 安装依赖项

撰写本文时，安装 Zephyr 有四项要求：

- CMake 版本 3.20.5 或者更高版本
- Python 版本 3.10 或者更高版本
- 器件树编译器版本 1.4.6 或者更高版本
- Git

### 3.1.2 设置 Python 及 Zephyr

Zephyr 有相当多的 Python 依赖项，务必始终更新这些依赖项。首次设置时，请执行以下步骤：

1. 使用以下命令在 Ubuntu 内安装依赖项：
```
sudo apt install --no-install-recommends git cmake ninja-build gperf \
  ccache dfu-util device-tree-compiler wget python3-dev python3-venv python3-tk \
  xz-utils file make gcc gcc-multilib g++-multilib libsdl2-dev libmagic1
```
2. 在 Home Path 内创建新的虚拟环境。每个新的 Zephyr 工程都应执行此操作。本示例将使用“zephyrproject”，但只要将来的任何命令也更新，就可以自由更改此名称。
```
python -m venv ~/zephyrproject/.venv #create script for virtual environment
source ~/zephyrproject/.venv/bin/activate #activate virtual environment
```
3. 在 zephyrproject 内部初始化 west 并 cd 到其中。运行 west 更新，这可能需要大约 15 分钟，因为需要安装许多程序包。
```
pip install wheel
pip install west
west init ~/zephyrproject
cd ~/zephyrproject
west update
west zephyr-export
west packages pip --install
```
4. 最后，安装 Zephyr SDK。
```
deactivate
cd ~
git clone https://github.com/openocd-org/openocd.git
sudo apt install libusb-1.0-0-dev libhidapi-dev
```

### 3.1.3 OpenOCD

Zephyr 包含 Zephyr 0.17.4 的 OpenOCD 版本，不支持 TI LaunchPad。因此，安装支持 TI MSP MCU 的较新版本至关重要。

如果本指南已经跟进到这一点，那么在主目录中为 OpenOCD 创建新文件夹之前，必须首先停用到目前为止所使用的虚拟环境。

```
deactivate

cd ~

git clone https://github.com/openocd-org/openocd.git

sudo apt install libusb-1.0-0-dev libhidapi-dev
```

完成此操作后，可在 OpenOCD 文件夹中构建 OpenOCD。对于 TI 硬件上的所有构建，在刷写过程中必须参考此构建。

```
cd <cloned_OPENOCD_dir>

git submodule update --init --recursive

cd jimtcl

./configure

make

sudo make install

cd .. #back in the cloned directory

sudo apt-get install libusb-1.0-0-dev

./bootstrap #when building from the git repository

./configure --enable-xds110 #optionally add any other debuggers as needed

make

sudo make install
```

OpenOCD 现在已可执行以下步骤。

### *3.1.4 让 TI 下游与众不同*

最后步骤是设置 TI 下游 Zephyr 存储库。TI 拥有由社区维护的 Zephyr 上游的单独分支。这称为 TI 下游，即 TI 维护的分支，其中包括尚未包含在上游中的功能。这包括其他电路板、用户特定的应用程序、新驱动程序等。此外，由于 TI 主动维护此分叉，用户可以使用 E2E 表单进行支持。对于上游和下游都存在的项目，建议使用上游分支。

第一步是检查分支的当前状态，查看远程指针的当前名称。

```
cd ~/zephyrproject/zephyr

git status

git remote -v
```

如果执行了前面的步骤，则会安装并跟踪上游。上述命令的输出应当如下所示：

```
origin  https://github.com/zephyrproject-rtos/zephyr (fetch)

origin  https://github.com/zephyrproject-rtos/zephyr (push)
```

Origin 是一个非特定名称，因此为了清晰起见，建议明确引用上游和下游。以下命令都将 Zephyr 原点重命名为上游，同时为 TI 的下游创建源。

```
git remote rename origin upstream

git remote add downstream https://github.com/TexasInstruments/msp-zephyr.git
```

运行 git remote -v 后的预期输出应如下所示：

```
#expected output after git remote -v
downstream      https://github.com/TexasInstruments/msp-zephyr.git (fetch)
downstream      https://github.com/TexasInstruments/msp-zephyr.git (push)
upstream        https://github.com/zephyrproject-rtos/zephyr (fetch)
upstream        https://github.com/zephyrproject-rtos/zephyr (push)
```

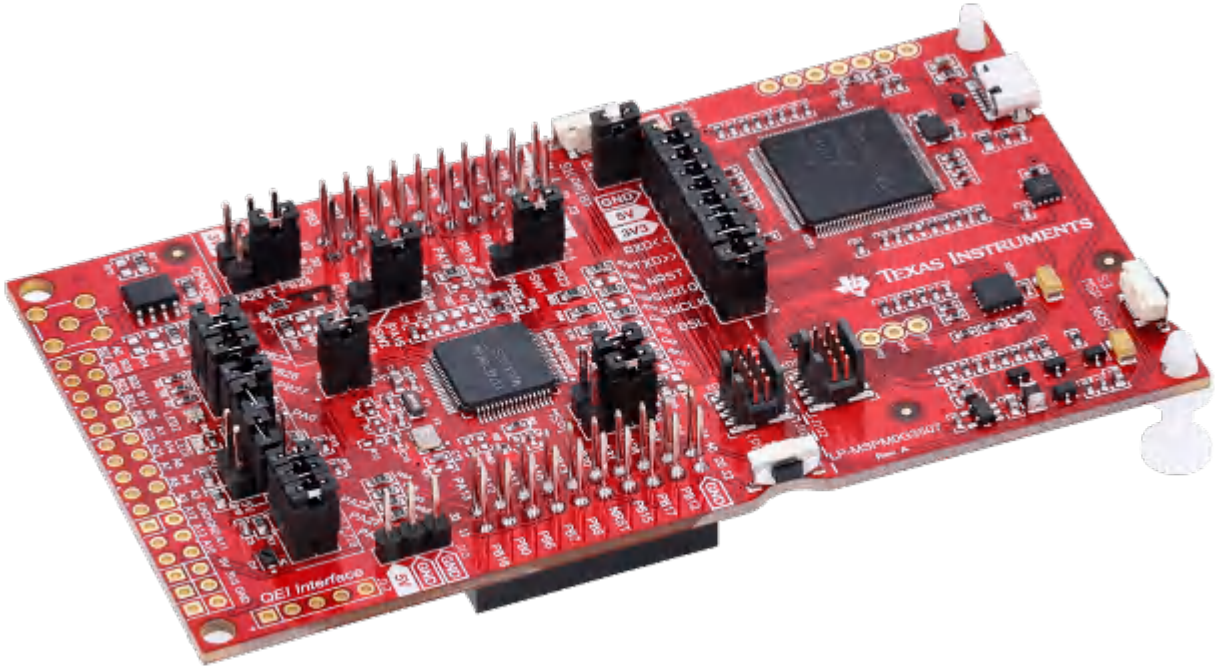
然后，可使用下游指针访问对下游的任何更新。下面是在开发新项目时签出现有分支并为版本管理创建新本地分支的示例。

```
git fetch downstream # receives information about the downstream

git checkout downstream/stable # checks out remote downstream stable branch without creating local branch

git checkout -b stable downstream/stable # creates new local branch called stable that tracks the downstream stable branch
```

# 4 如何在 MSPM0 Launchpad 上运行示例

## 4.1 MSPM0 Launchpad

MSPM0 系列器件具有名为 Launchpad 的评估模块，可用于启动并运行 Zephyr 测试和原型设计。下面是 LP-MSPM0G3507，将用于以下教程。




## 4.2 在 MSPM0 Launchpad 上运行工程

以下各节基于 Zephyr 文档中的入门指南。但是当发生重要变化时，我们将在此处提及这些变化。

### *4.2.1 运行 Blinky*

正确设置文件系统后，通过 USB 连接 LaunchPad。在虚拟环境中，重新进入虚拟环境，cd 到 /zephyr 文件夹中，然后运行以下命令：

```
cd ~/zephyrproject/zephyr
```

```
west build -p always -b lp_mspm0g3507 -d out samples/basic/blinky
```

```
west flash -d out --openocd ~/openocd/src/openocd --openocd-search ~/openocd/tcl
```

这会使用之前安装的 OpenOCD 版本构建并且刷写电路板。

*若要查看当前支持的电路板，请运行 west build，并使用与 LaunchPad 相对应的电路板名称。对于 LP-MSPM0G3507 LaunchPad™，电路板名称是 lp_mspm0g3507。*

当刷写完成时，LED 将在硬件重新启动时定期闪烁。刷写时，电路板会进入调试模式，并允许使用 west 调试命令或第 5.2.3 节“调试工程”中所述的其他调试工具进行调试。

### *4.2.2 运行更为复杂的示例*

有更多涵盖其他 MSPM0 外设的复杂示例，使用户能够使用这些预编译示例快速进行原型设计，或修改它们以用作更大、更复杂工程的起点。

与 blanky 类似，要立即运行这些示例，west build 将编译工程，然后可使用 west flash 将工程刷写到电路板上。Zephyr 包含为不同电路板构建的各种样片，并且由于 Zephyr 的便携式设计，有许多样片可轻松移植到 MSPM0。这些示例可以在样本/文件夹中找到，并且具有不同的复杂度。

## 4.3 调试工程

### *4.3.1 带有命令行的 GNU 调试程序 (GDB)*

GDB 是一个命令行界面，可用于连接 XDS-110 仿真器以调试 Zephyr 项目。与 CCS 类似，编译 Zephyr 工程时创建的 Zephyr.elf 文件允许使用 GDB 进行调试。

west 调试有多种选项；但是，建议读者阅读 Zephyr 工程的 west 调试指南，以了解他们可能使用的任何特定运行器或调试工具。

### *4.3.2 设置 Visual Studio Code (VSCode) 环境*

Visual Studio Code 是调试嵌入式项目的常用方法，具有正确的扩展。

利用扩展 Cortex-Debug，可以在 Visual Studio Code 内进行可视调试。通过 VSCode 中的“extensions menu”安装扩展后，必须配置 launch.json 文件以支持 MSPM0 Zephyr 工程。之后，环境就可以构建、刷写和调试 Zephyr 工程了。

```
{
    "version": "2.0.0",
    "configurations": [
        {
            "name": "Zephyr Debug",
            "executable":"<absolute_path_to>/zephyrproject/zephyr/out/zephyr/zephyr.elf,
            "request": "launch",
            "type": "cortex-debug",
            "runToEntryPoint": "main",
            "servertype": "external",
            "gdbPath": "<absolute_path_to>/<path_to_zephyr_sdk>/arm-zephyr-eabi/bin/arm-
zephyr-eabi-gdb",
            "gdbTarget": "localhost:3333",
            "device": "MSPM0G3507"
        },
    ]
}
```

### *4.3.3 在 VSCode 中使用 Cortex-Debug 进行调试*

在 VSCode 中，使用电路板的 .ccxml 连接到电路板（可以在大多数示例中的“目标配置”部分下找到），然后使用它开始无工程调试。然后，使用 west build 命令时由 west 生成的 Zephyr.elf 可用于加载符号。

假设已使用 debug-cortex 设置 VSCode 环境（如节 4.3.2 所示），则可以使用以下命令启动调试服务器并在 VSCode 中运行开始调试。

```
west debugserver -d out --openocd ~/openocd/src/openocd \
--openocd-search ~/openocd/tcl
```

### 4.4 创建自己的工程

可以根据现有示例构建新工程，也可以通过在 zephyrproject/zephyrfolder（存储库应用）、zephyrproject/文件夹本身（工作区应用）中创建新文件夹，或在之前完全创建的文件夹（独立应用）之外创建新文件夹，由此来创建新工程。可在 Zephyr 文档中找到有关这些不同应用程序类型的更多信息。

为了创建独特的工程，Zephyrproject 的应用程序开发部分涵盖了需要了解的所有内容。一般来说，基于 TI 或 Zephyr 提供的某个示例创建工程是创建工程的更简单方法，因为这些示例默认包含必要的配置文件，这意味着无需过多地处理覆盖层、prj.conf 文件等。

# 5 参考资料

- RTOS 基础知识指南
- LP-MSPM0G3507 LaunchPad
- Zephyr 文档入门指南
- Cortex-Debug VSCode 扩展

# 6 E2E

如需 TI 产品支持，请使用 E2E 支持表单查看常见问题解答并发布新问题。

# 7 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

| 日期 | 修订版本 | 注释 |
| --- | --- | --- |
| 2026 年 2 月 | * | 初始发行版 |

# 重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。



最后更新日期：2025 年 10 月